Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №2**

по дисциплине: «Методы оптимизации».

Вариант - 8

Выполнил:

студент 1 курса, гр. ИВТАСмд-11

Кондратьев Павел Сергеевич.

Проверил:

преподаватель кафедры ВТ

Валюх Вероника Валерьевна

г. Ульяновск, 2020

Оглавление

[Задание: 3](#_Toc55840920)

[Ход выполнения работы: 4](#_Toc55840921)

[Описание метода метод наискорейшего спуска 4](#_Toc55840922)

[Реализация и сравнение эффективности программной реализации алгоритма 9](#_Toc55840923)

[Онлайн реализации метода наискорейшего спуска 14](#_Toc55840924)

[Исследование реализации выбранного алгоритма в различных пакетах прикладных программ (MathCad; MATLAB). 15](#_Toc55840925)

[Список источников: 17](#_Toc55840926)

# Задание:

1. Составление библиотеки ссылок на исходники программ и онлайн реализации метода. Исследование реализации выбранного алгоритма в различных пакетах прикладных программ (Maple; MathCad; Mathematica; MATLAB)

2. Осмысление и описание метода оптимизации.

3. Реализация и сравнение эффективности реализации алгоритма в разных языках программирования (С++, С#, JAVA, язык по выбору). Исследование зависимости скорости выполнения от а) размерности задачи, б) языка.

4. Оформить отчёт о проделанной работе.

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Лабораторная 1**  **Методы поиска для функции одной переменной (одномерная оптимизация)**  **Методы последовательного поиска** | **Лабораторная 2**  **Методы оптимизации дифференцируемых функций**  **(Градиентные методы, прямые методы, методы первого порядка) \*** | **Лабораторная 3**  **Оптимизация на графах**  **(Поиски кратчайших путей, построение деревьев)** |
| 8 | метода Ньютона | Модифицированный метод наискорейшего спуска | Алгоритм Флойда |

**Лабораторная №2.**

# Ход выполнения работы:

## Описание метода метод наискорейшего спуска

Метод наискорейшего спуска (в англ. литературе «method of steepest descent») - это итерационный численный метод (первого порядка) решения оптимизационных задач, который позволяет определить экстремум (минимум или максимум) целевой функции:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image001.png

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image002.png - это значения аргумента функции (управляемые параметры) на вещественной области.

В соответствии с рассматриваемым методом экстремум (максимум или минимум) целевой функции определяют в направлении наиболее быстрого возрастания (убывания) функции, т.е. в направлении градиента (антиградиента) функции. Градиентом функции  в точке http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image002.png называется вектор, проекциями которого на координатные оси являются частные производные функции по координатам:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image003.png

где i, j,…, n - единичные векторы, параллельные координатным осям.

Градиент в базовой точке http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image004.png строго ортогонален к поверхности, а его направление показывает направление наискорейшего возрастания функции, а противоположное направление (антиградиент), соответственно, показывает направление наискорейшего убывания функции.

Метод наискорейшего спуска является дальнейшим развитием метода градиентного спуска. В общем случае процесс нахождения экстремума функции является итерационной процедурой, которая записывается следующим образом:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image005.png

где знак «+» используется для поиска максимума функции, а знак «-» используется для поиска минимума функции;

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image006.png- единичный вектор направления, который определяется по формуле:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image007.png

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image008.png- модуль градиента определяет скорость возрастания или убывания функции в направлении градиента или антиградиента:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image009.png

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image010.png- константа, определяющая размеры шага и одинаковая для всех i-х направлений.

Величина шага выбирается из условия минимума целевой функции f(х) в направлении движения, т. е. в результате решения задачи одномерной оптимизации в направлении градиента или антиградиента:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image011.png

Другими словами, величину шага   определяют при решении данного уравнения:

http://simenergy.ru/MyArticles/Math_analysis_Solution_methods/003/003_steepest_descent.files/image013.png

Таким образом, шаг расчета выбирается такой величины, что движение выполняется до тех пор, пока происходит улучшение функции, достигая, таким образом, экстремума в некоторой точке. В этой точке вновь определяют направление поиска (с помощью градиента) и ищут новую точку оптимума целевой функции и т.д. Таким образом, в данном методе поиск происходит более крупными шагами, и градиент функции вычисляется в меньшем числе точек.

В случае функции двух переменных данный метод имеет следующую геометрическую интерпретацию: направление движения из точки *xk* касается линии уровня в точке *xk+1*. Траектория спуска зигзагообразная, причем соседние звенья зигзага ортогональны друг другу.

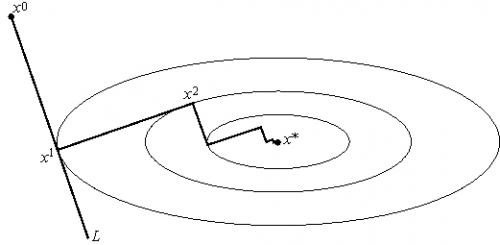
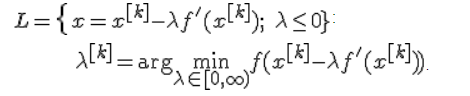


Рис.1 Геометрическая интерпретация метода наискорейшего спуска. На каждом шаге выбирается так, чтобы следующая итерация была точкой минимума функции на луче L.

Этот вариант градиентного метода основывается на выборе шага из следующего соображения. Из точки x[k] будем двигаться в направлении антиградиента до тех пор, пока не достигнем минимума функции f на этом направлении, т. е. на луче:



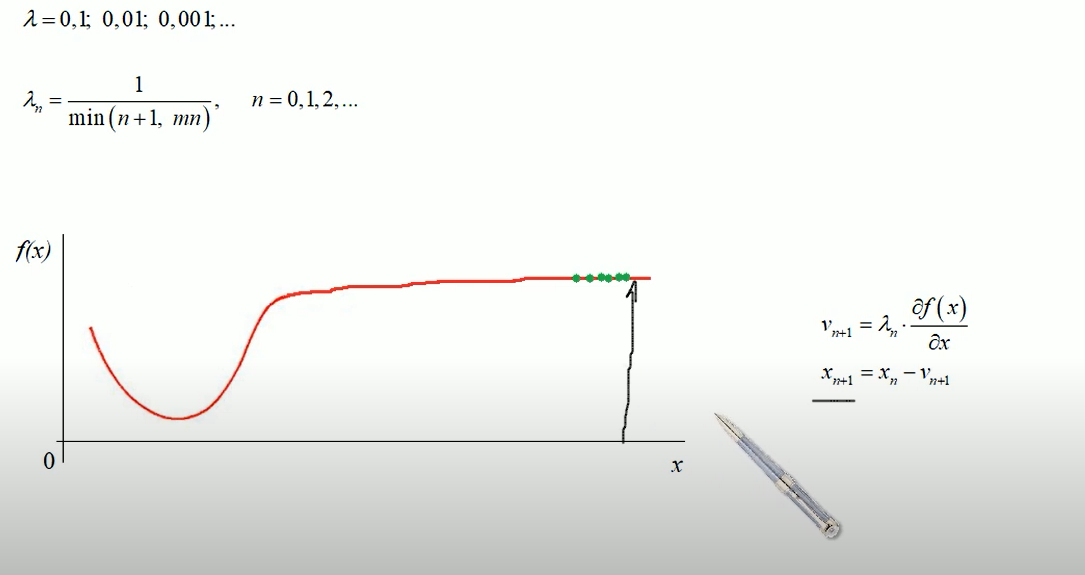
Другими словами, L[k] выбирается так, чтобы следующая итерация была точкой минимума функции f на луче L (см. рис. 1). Такой вариант градиентного метода называется методом наискорейшего спуска. Заметим, кстати, что в этом методе направления соседних шагов ортогональны.

Метод наискорейшего спуска требует решения на каждом шаге задачи одномерной оптимизации. Практика показывает, что этот метод часто требует меньшего числа операций, чем градиентный метод с постоянным шагом.

В общей ситуации, тем не менее, теоретическая скорость сходимости метода наискорейшего спуска не выше скорости сходимости градиентного метода с постоянным (оптимальным) шагом.

**Методы оптимизации**

1. Избежать попадания в локальный минимум
2. Обеспечить наибольшую скорость сходимости
3. Для решения одномерных задач оптимизации использован метод **золотого сечения**
4. Сходимость градиентного спуска с постоянным шагом
5. Градиентный метод с дроблением шага
6. Изменения шага сходимости



1. Adagrab – учитывает при оптимизации квадраты градиентов
2. RMSProp и Adadelta – подобный Adagrab, но пытаются бороться с чрезмерным накоплением квадратов градиентов
3. Adam – смесь алгоритма с моментом и квадратом градиентов

**Недостатком** методом **наискорейшего спуска** является необходимость решать одномерную задачу оптимизации.

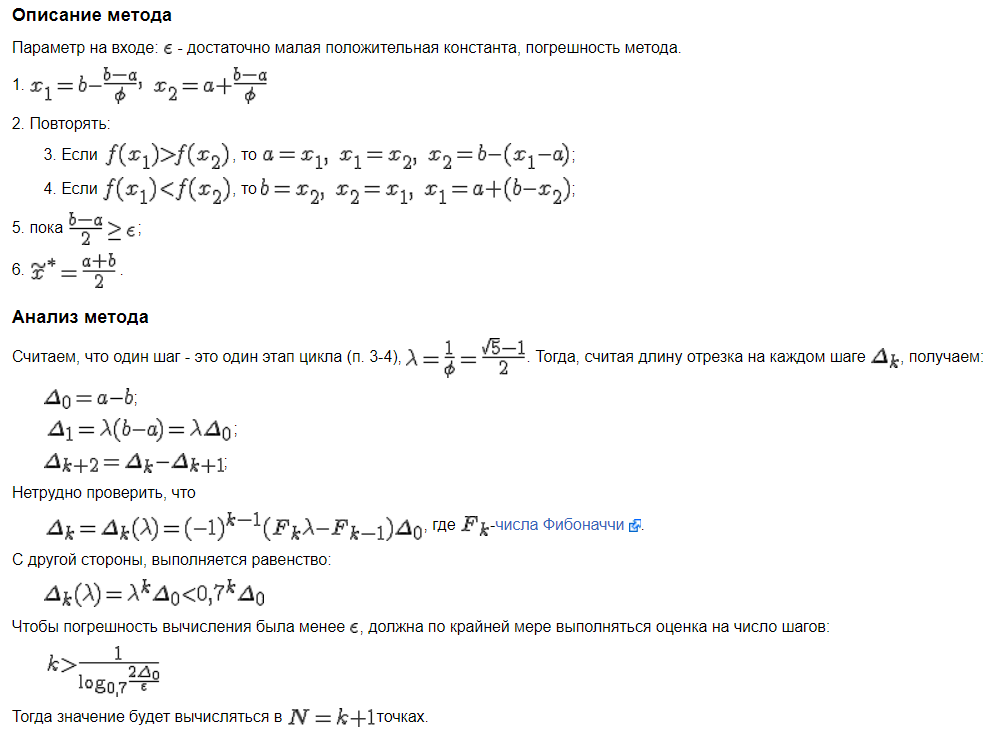
**Рекомендации**

При программировании методов градиентного спуска следует аккуратно относится к выбору параметров, а именно

Метод градиентного спуска с постоянным шагом: шаг lambda следует выбирать меньше 0.01, иначе метод расходится (метод может расходится и при таком шаге в зависимости от исследуемой функции).

Градиентный метод с дроблением шага не очень чувствителен к выбору параметров. Один из вариантов выбора параметров: eps =0.1, delta=0.95, lambda^0=1

Метод наискорейшего спуска: в качестве метода одномерной оптимизации можно использовать метод золотого сечения (когда он применим).



**Недостатки**

Неустойчивость относительно ошибок округления: мы получаем приблизительные значения чисел phi и lambda, дальнейшие вычисления только накапливают ошибки, что может привести к нарушению условия вложенности отрезков a\_k, b\_k и расходимости процесса.

Методы градиентного спуска являются достаточно мощным инструментом решения задач оптимизации. Главным недостатком методов является ограниченная область применимости.

# Реализация и сравнение эффективности программной реализации алгоритма

Составим программу для поиска максимума и минимума на некотором интервале для следующей функции: 1.5 \* sin(1.5 \* pow(x, 2) + 3).

Реализация на C++

using System;

using System.Diagnostics;

namespace lab2

{

class Program

{

static double func(double x) // функция вызова функции

{

return 1.5 \* Math.Sin(1.5 \* Math.Pow(x, 1) + 3);

//return (10 \* x \* log10(x) / log10(2.7) - (x \* x) / 2); // заданная задачей функция

}

static void mZolotSech(double eps, double a, double b) // сам метод

{

double x1, x2, y1, y2, ymin, xmin, t, er, ea; // er -> расчетная погрешность, еа->вычисляемая погрешность.

int m, j;

t = (Math.Sqrt(5) - 1) / 2; // то самое phi, отношение золотого сечения

x1 = b + (b - a) / t; y1 = func(x1);

x2 = a - (b - a) / t; y2 = func(x2);

m = 2;

while ((b - a) > eps)

{

if (y1 < y2)

{

b = x2;

x2 = x1; y2 = y1;

x1 = b - (b - a) \* t;

y1 = func(x1);

}

else

{

a = x1;

x1 = x2; y1 = y2;

x2 = a + (b - a) \* t;

y2 = func(x2);

}

m++;

}

if (y1 < y2) b = x2;

else a = x1;

xmin = (a + b) / 2.0; ymin = func(xmin); ea = (b - a) / 2.0;

}

static void Main(string[] args)

{

Console.Write("Введите a = ");

double a = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите b = ");

double b = Convert.ToDouble(Console.ReadLine());

Stopwatch sw = new Stopwatch();

for (double eps = 0.1; eps >= 0.0000000001; eps \*= 0.1)

{

sw.Start(); // стартовая засечка

for (int i = 0; i < 1000; i++)

mZolotSech(eps, a, b);

sw.Stop(); // замер продолжительности

Console.WriteLine($"{sw.ElapsedMilliseconds} мс {eps} eps");

}

}

}

}

Реализация на C++

#include <iostream>

#include <math.h>

#include <conio.h>

#include <intrin.h> // там \_\_rdtsc()

#include <iomanip>

using namespace std;

double func(double x) // функция вызова функции

{

return 1.5 \* sin(1.5 \* pow(x, 1) + 3);

//return (10 \* x \* log10(x) / log10(2.7) - (x \* x) / 2); // заданная задачей функция

}

void mZolotSech(double eps, double a, double b) // сам метод

{

double x1, x2, y1, y2, ymin, xmin, t, er, ea; // er -> расчетная погрешность, еа->вычисляемая погрешность.

int m, j;

t = (sqrtl(5) - 1) / 2; // то самое phi, отношение золотого сечения

//er = (b - a) / (2 \* pow(t, (N - 1)));

x1 = b + (b - a) / t; y1 = func(x1);

x2 = a - (b - a) / t; y2 = func(x2);

m = 2;

while ((b - a) > eps)

{

if (y1 < y2)

{

b = x2;

x2 = x1; y2 = y1;

x1 = b - (b - a) \* t;

y1 = func(x1);

//cout << "m=" << m << " x1=" << x1 << " x2=" << x2 << " y1=" << y1 << " y2=" << y2 << endl;

}

else

{

a = x1;

x1 = x2; y1 = y2;

x2 = a + (b - a) \* t;

y2 = func(x2);

//cout << "m=" << m << " x1=" << x1 << " x2=" << x2 << " y1=" << y1 << " y2=" << y2 << endl;

}

m++;

}

// m < N

if (y1 < y2) b = x2;

else a = x1;

//cout << endl;

//cout << a << b;

xmin = (a + b) / 2.0; ymin = func(xmin); ea = (b - a) / 2.0;

//cout << endl;

//cout << setw(15) << xmin << setw(15) << ymin << setw(15) << er << setw(15) << ea << endl;

}

// частота в мегагерцах через RDTSC

int mhz\_cpu() {

clock\_t clock\_tick1, clock\_tick2;

\_\_int64 cpu\_tick1, cpu\_tick2;

int rdtsc\_tick;

double usec; // время в микросекундах

clock\_tick1 = clock();

while ((clock\_tick2 = clock()) == clock\_tick1); // пропуск остатка текущего тика clock

cpu\_tick1 = \_\_rdtsc(); // взять TSC

while (clock() == clock\_tick2); // отсчет одного тика clock

cpu\_tick2 = \_\_rdtsc() - cpu\_tick1; // сколько натикал счетчик TSC за один тик clock

// вычисляем частоту в мегагерцах

usec = 1000000.0 / CLOCKS\_PER\_SEC; // время одного тика clock в микросекундах

return int(cpu\_tick2 / usec);

}

int main()

{

clock\_t clock\_start, clock\_time; // стартовое и измеренное время для clock()

\_\_int64 hpet\_start, hpet\_end, hpet\_time, // стартовое, финишное и измеренное время для HPET

hpet\_freq; // частота HPET

\_\_int64 rdtsc\_start, rdtsc\_time, // стартовое, финишное и измеренное время для RDTSC

rdtsc\_freq; // частота счетчика тактов (процессора) в мегагерцах

setlocale(LC\_ALL, "Russian");

cout << "Добро Пожаловать в программу\n";

double a, b, eps;

cout << "\nВведите a="; cin >> a; cout << "\nВведите b="; cin >> b;

for (double eps = 0.1; eps > 0.0000000001; eps \*= 0.1)

{

rdtsc\_start = \_\_rdtsc();

mZolotSech(eps, a, b);

rdtsc\_time = \_\_rdtsc() - rdtsc\_start;

cout << rdtsc\_start << ": " << rdtsc\_time << " тактов";

rdtsc\_freq = mhz\_cpu();

cout << "\nВремя = " << double(0.001 \* rdtsc\_time / rdtsc\_freq) << " мсек ";

}

}

Реализация на Python

# Подключение библиотек

import time

import math

import pylab

import matplotlib.pyplot as plt

from matplotlib import mlab

# Определение переменных

A = 1.5

B = 1

C = 3

D = 1.5

x\_min = 2.2

x\_max = 4.4

# Определение функции

function\_f = lambda x: D \* math.sin(A \* x \*\* B + C)

# Метод золотого сечения

def Golden\_Section\_Method(x\_min, x\_max, eps):

iteration = 1.0

#print((" {0:.8s} || {1:.5s} || {2:.8s} || {3:.5s} || {4:.8s}").format("Итерация", "x\_min", "f(x\_min)", "x\_max",

# "f(x\_max)"))

coefficient = (math.sqrt(5) - 1) / 2

d = x\_min + (x\_max - x\_min) \* coefficient

c = x\_max - (x\_max - x\_min) \* coefficient

sc = function\_f(c)

sd = function\_f(d)

while (x\_max - x\_min) > eps:

if (sd < sc):

x\_max = d

d = c

c = x\_max - (x\_max - x\_min) \* coefficient

sd = sc

sc = function\_f(c)

else:

x\_min = c

c = d

d = x\_min + (x\_max - x\_min) \* coefficient

sc = sd

sd = function\_f(d)

iteration += 1

#print((" {0:.0f} || {1:.4f} || {2:.4f} || {3:.4f} || {4:.4f}").format(iteration - 1, x\_min,

# function\_f(x\_min), x\_max,

# function\_f(x\_max)))

start\_time = time.time()

for j in range(1000):

Golden\_Section\_Method(x\_min, x\_max, 0.0000000001)

print(time.time() - start\_time)

# Шаг между точками

dx = 0.1

# Создадим список координат по оси X на отрезке [-x\_min; x\_max], включая концы

xlist = mlab.frange(x\_min, x\_max, dx)

# Вычислим значение функции в заданных точках

ylist = [function\_f(x) for x in xlist]

# Нарисуем одномерный график

pylab.plot(xlist, ylist)

plt.grid(True)

# Покажем окно с нарисованным графиком

pylab.show()

Исследуем зависимость скорости выполнения программного кода от

1. а) заданной погрешности вычисления
2. б) языка программирования

f(x) = 1.5 \* sin(1.5 \* pow(x, 2) + 3)

|  |  |  |  |
| --- | --- | --- | --- |
| Погрешность вычисления, | Время выполнения программного кода, мс | | |
| C# | C++ | Python |
| 0.01 | 0.0035112 | 0.0024051 | 0.0069794 |
| 0.001 | 0.0093524 | 0.0028104 | 0.0099730 |
| 0.0001 | 0.01101502 | 0.00301549 | 0.0119700 |
| 0.00001 | 0.01358044 | 0.00377083 | 0.0149610 |
| 0.000001 | 0.01310308 | 0.00307117 | 0.0179162 |
| 0.0000001 | 0.01426387 | 0.00232338 | 0.0219428 |
| 0.00000001 | 0.01762351 | 0.00196276 | 0.0238893 |
| 0.000000001 | 0.01944476 | 0.00284476 | 0.0259318 |
| 0.0000000001 | 0.02113724 | 0.00231674 | 0.0312489 |

Анализ охватывает следующие средства:

1. **Язык С++ - режим Debug, функция Clock(), использование команды RDTSC**

На всех UNIX-подобных ОС, очень старая функция clock( ) возвращает процессорное время процесса в тиках, а макрос CLOCKS\_PER\_SEC количество тиков в секунду.

Функция clock() описывается в time.h и возвращает число тиков от момента загрузки программы. Тик обычно равен 1 миллисекунде, но для возможности в последующем работать с другой длительностью такта в time.h фиксируется константа CLOCKS\_PER\_SEC (время изерения было взять в нс).

Счетчик тактов процессора TSC (Time Stamp Counter) встроен в ядро CPU и обычно прирастает на 1 с каждым тактом задающего генератора (это тот, что определяет гигагерцы процессора). В С/С++ он используется через функцию \_\_int64 \_\_ rdtsc(), описанную в windows.h.

1. **С# режим Debug, Stopwatch Класс**

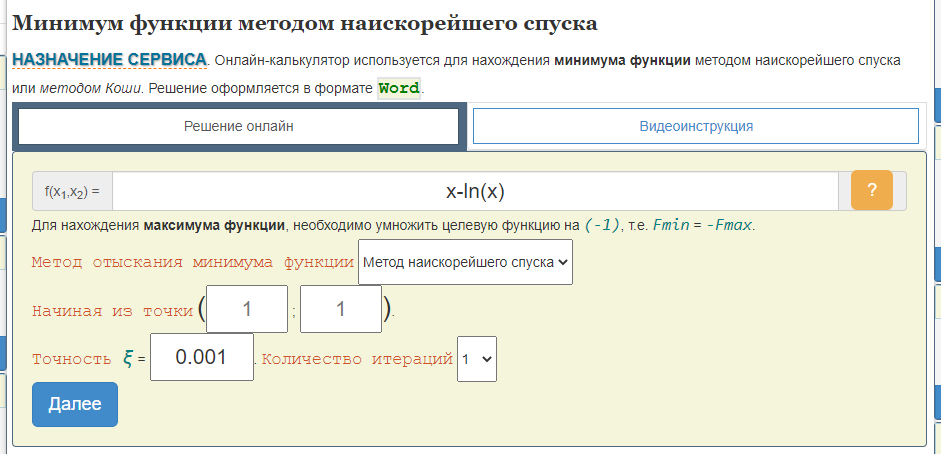
Класс Stopwatch основан на HPET (High Precision Event Timer, таймер событий высокой точности). Данный таймер был введён фирмой Microsoft, чтобы раз и навсегда поставить точку в проблемах измерения времени. Частота этого таймера (минимум 10 МГц) не меняется во время работы системы. Для каждой системы Windows сама определяет, с помощью каких устройств реализовать этот таймер.

На многопроцессорном компьютере не имеет значения, на каком процессоре выполняется поток. Однако из-за ошибок в BIOS или слое абстрагирования оборудования (HAL) можно получить разные временные результаты на разных процессорах. Чтобы указать соответствие процессоров для потока, используйте ProcessThread.ProcessorAffinity метод.

По результатам экспериментов можно сделать вывод, что скорость работы программ на языке C++ выше, чем скорость работы программ, реализующих метод наискорейшего спуска на языках C# и Python. Это связано с тем, что С++ компилируется непосредственно в машинный код и работает с минимально возможным количеством хелперов и прослоек. Шаблоны C# определяются во время выполнения и это медленнее, чем шаблоны времени компиляции C++.

# Онлайн реализации метода наискорейшего спуска

На сайте <https://math.semestr.ru> реализован поиск минимума функции методом наискорейшего спуска. Необходимо перейти по ссылке: <https://math.semestr.ru/optim/steepest-descent.php>, выбрать в списке метод наискорейшего спуска, задать функцию, задать точность вычисления и нажать кнопку Далее.



Исходники программ для реализации метода наискорейшего спуска на языке **С#** представлены на сайте <https://studassistent.ru>. Для перехода к исходникам используем ссылку: <https://studassistent.ru/charp/metod-naiskoreyshego-spuska-c>.

# Исследование реализации выбранного алгоритма в различных пакетах прикладных программ (MathCad; MATLAB).

Готовых реализаций с помощью средств пакета MathCAD не получилось выявить.

Реализация метода наискорейшего спуска для нахождения минимума функции в Matlab.

Метод наискорейшего спуска для нахождения минимума функции:

clear;

clc;

tic

format long

% Initial values

a\_lower = 1; % lower limit

b\_upper = 2; % upper limit

ro = (3 - sqrt(5))/2; % constant step size (golden number)

eps = b\_upper - a\_lower;

% Assign initial value

a\_old = a\_lower;

b\_old = b\_upper;

iter = 0; %

% Golden Section Algorithm

func = @(x) x^2 + 4\*cos(x); % you can change the

while(eps > 10^-5)

a\_new = a\_old + (b\_old - a\_old)\*ro; % new lower limit

b\_new = b\_old - (b\_old - a\_old)\*ro; % new upper limit

if (func(b\_new) > func(a\_new))

b\_old = b\_new;

elseif (func(b\_new) < func(a\_new))

a\_old = a\_new;

end

eps = b\_old - a\_old;

iter = iter + 1;

end

timeElapsed = toc;

fprintf("\tGolden Search Method\n")

fprintf("Iteration : %d\nAverage computation time : %f\n",iter,timeElapsed);

fprintf("Minimum point of function : x = %f, f(x) = %f\n",a\_new,func(a\_new));

**Описание**

Локальные минимизатор следующей функции:

f(x) = x^2 + 4cos(x)

Код поиска локального минимизатора этой функции на интервале [1,2].

Начальная точка дана следующая. x = 1

Чтобы запустить эти коды, просто напишите в командной строке в Matlab

run anysearchalgorithm.m

Вывод будет выглядеть следующим образом

Secant Method

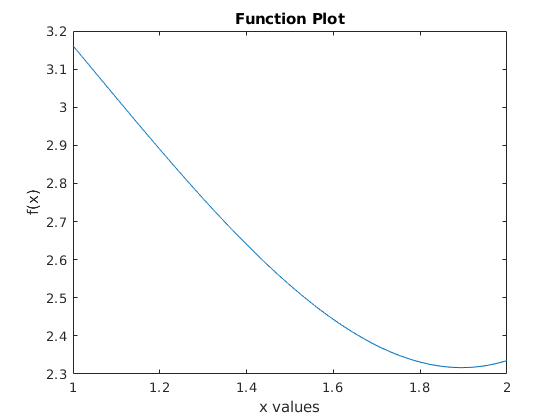
Iteration : 19

Computation time : 0.008753

Minimum point of function : x = -1.895494, f(x) = 2.316808

**Вывод**

Функция имеет только одну локальную минимальную точку функции на этом интервале. Если функция не является унимодальной, то поисковые алгоритмы могут застрять в любом локальном минимизаторе функции. График функции:



# Список источников:

1. Аттетков А.В., Галкин С.В., Зарубин B.C. МЕТОДЫ ОПТИМИЗАЦИИ Учеб. для вузов /Под ред. B.C. Зарубина, А. П. Крищенко. - 2-е изд., стереотип. - М.: МГТУ им. Н. Э. Баумана, 2003.
2. Маркина М.B., Судакова А.В. ПРАКТИКУМ ПО РЕШЕНИЮ ЗАДАЧ ОПТИМИЗАЦИИ В ПАКЕТЕ MATLAB: учебно-методическое пособие. –Нижний Новгород: Нижегородский госуниверситет, 2017.
3. Прокопенко Н. Ю. Методы оптимизации: учеб. пособие /Н. Ю. Прокопенко; Нижегор. гос. архитектур. - строит. ун-т. – Н. Новгород: ННГАСУ, 2018.
4. Банди Б. Методы оптимизации. Вводный курс.: Пер. с англ. – М.: Радио и связь, 1988. – 128 с.
5. Метод градиентного спуска - [http://www.machinelearning.ru/wiki/ index.php?title=Метод\_градиентного\_спуска](http://www.machinelearning.ru/wiki/%20index.php?title=Метод_градиентного_спуска) (дата обращения 07.11.2020)